Defining and Understanding Software Measurement Data[1]

James A. Rozum
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-7770
jar@sei.cmu.edu

Abstract. For managers to effectively manage and control software development projects, they need to incorporate a measurement process into their decision making and reporting process. Measurement costs money, but it can also save money through early problem detection and objective clarification of critical software development issues. The following describes a measurement process and provides some basic concepts that managers can use to help integrate measurement into the process for managing software development.

Keywords: Software measurement, metrics, data collection, software process, measurement process, process improvement.

Why a Process for Measurement

"it's not so much what I don't know, it's what I know that's not so" Dooley

Dooley could never have been more right. Projects collect lots of data—most of it they don't recognize as data. There is still more data available that projects don't realize exist, or haven't considered using the data as information. There is little project managers don't know about their projects, yet what always seems to cause problems are those things that projects thought they had under control. According to Deming, information is used to understand theory. Knowledge is the result of that understanding [Deming]. The problems encountered by project managers is that often their theory has been proven incorrect by information. The project manager's theory is the plan including the schedule, budget, and expected quality of the resulting product. The information being received by the project manager are actually signals that their project is going to be late, cost more than expected, or not work quite the way it was originally planned; sometimes all three signals are received.

Why does this happen? Oftentimes it is because of theory that is based on poor information. A defined measurement process that is integrated into the software development process will provide the reliable information and insight needed by managers to make better decisions, i.e., their theories. With a defined software measurement process, managers have reliable data and information that provides early insight into potential problems and can support

---

[1] This paper has been previously published in the proceedings from the 5th International Applications of Software Measurement Conference.

 This Work is sponsored by the U.S. Department of Defense

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JAN 2001		**00-00-2001 to 00-00-2001**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Defining and Understanding Software Measurement Data	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Software Engineering Institute,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
For managers to effectively manage and control software development projects, they need to incorporate a measurement process into their decision making and reporting process. Measurement costs money, but it can also save money through early problem detection and objective clarification of critical software development issues. The following describes a measurement process and provides some basic concepts that managers can use to help integrate measurement into the process for managing software development.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**11**	

decisions. Without a measurement process, managers do not have the information needed to identify and resolve problems. As a result, the managers react to events and problems rather than curtail those problems before they surface.

A software measurement process is a systematic method of measuring, assessing, and adjusting the software development process using objective data. Within such a systematic approach, software data is collected based on known or anticipated development issues, concerns, questions, or needs. The data are analyzed with respect to the characteristics of the software development process and products, and used to assess progress, quality, and performance throughout the development. There are four key components to an effective measurement process:

- Defining clearly the software development issues and the software measures (data elements) that support insight to the issues.
- Processing the software data into graphs and tabular reports (indicators) that support the analysis of issues.
- Analyzing the indicators to provide insight into the issues.
- Using the results to implement improvements and identify new issues and questions.

Each component of this process is driven by the issues and characteristics inherent to the program. For example, the decision of what specific data to collect is based on the list of issues to be addressed by measurement; the indicators developed from the measurement data are flexible and tailored by the development issues and related questions; and analysis techniques are chosen based on what information is desired and what question(s) need to be answered. Analysis can be directed at assessing the feasibility of development plans, identifying new issues, tracking issue improvement trends, projecting schedules based on performance to date, defining possible development tradeoffs, and evaluating the consistency and quality of development activities and products.

> "The data collection process must be driven by the information from questions that we formulate based on our needs. In short, know what question is to be answered before collecting data." [Juran]

The PDCA Measurement Process

The measurement process becomes a complete Plan-Do-Check-Act (PDCA) process that is integrated into the development process and the project manager's decision making process [Shewhart]. The PDCA cycle that a project can use is:

Plan: Identify the issues or questions the project and manager has and then determine the data and measures to be collected to address them.

Do: Collect data and, based on the issues identified, and using baseline data, derive graphical representations (i.e., indicators) to better illustrate the data trends.

Check: Analyze the trends, graphs, data, etc. to better understand the issues and the performance towards their resolutions.

Act: Report the results, recommend improvements, and identify new issues and questions.

As with the development of other system components, the key issues for software development relate to resource adequacy and expenditure, development progress and schedule, and the quality of the interim and final products. The nature of software development results in these three areas being strongly related [Rozum]. Issues with software quality, for example, can in many cases be directly attributable to schedule and resource constraints. This requires that the software measurement process address the relationships between several types of software data. For example, if a project is overrunning its planned allocations of staff hours, will the project be over the budget? Maybe, but if the rate of completing project activities is commensurate with the rate of higher than expected expenditures of staff hours, the project could finish within the budgeted number of staff hours, and complete earlier than expected.

The manager's issues related to the adequacy and expenditure of resources usually revolve around the adequacy of and accuracy of estimates and the availability and allocation of resources when they are needed.

The progress issue is simple; will the contract be completed on schedule, and if not, when will it be completed? However, factors that will affect determining this include: progress to date, amount of rework required, and the accuracy of earlier planning assumptions, e.g., productivity and project complexity.

Technical quality issues include both the quality of products being produced and the quality of the processes used to produce those products. Here the manager is concerned that interim products are stable and complete so that successive processes using those products do not compound mistakes (i.e., defects) and thereby drive up cost through added rework. Issues about the technical quality that managers typically have usually relate to defects, completeness of products, adequacy of processes, and stability and volatility of the system and its mission.

Collecting Data that Supports Insight to Issues

The question of what data to collect and how to define that data is driven by the project's current issues, projected issues, and characteristics of the software development process and products. The list of issues for measurement to provide insight to, is determined by the manager who identifies a list of project specific issues and concerns, sets priorities for the issues, and determine which of the issues can be addressed with the software measurement process. The manager can then determine what data is needed to support insight into the issues. However, not all issues can be predetermined or projected; therefore, the manager also needs to include provisions that allow the process to be flexible so that it can be modified to provide insight into unforeseen issues.

Change in the process is inevitable as the development proceeds and requirements become better defined: cost, performance, and schedule estimates are refined; personnel, machine,

test, and support resource requirements are identified; and insights from the measures themselves are gained. As the issues change, the manager should use the new information to adapt the measurement process to address the changes.

Once the data collection has started, measurement definitions should not be changed (e.g., lines of code definitions should not be changed). Changing a definition after the data collection has started will produce variations in the trends that could confuse the analyses and camouflage performance or related problems. However, sometimes definitions will change. When they do, it becomes critically important that the manager understands the change and how the change affects data that has already been collected. In Figure 1 for example, a new methodology was used to estimate the size of the project starting at month ten. This change in methodology produced a large variation in the estimated size. Not all changes will have such a dramatic impact. Still, the manager must understand the differences between the methods and the impacts those differences have on the decision making process [Rozum].

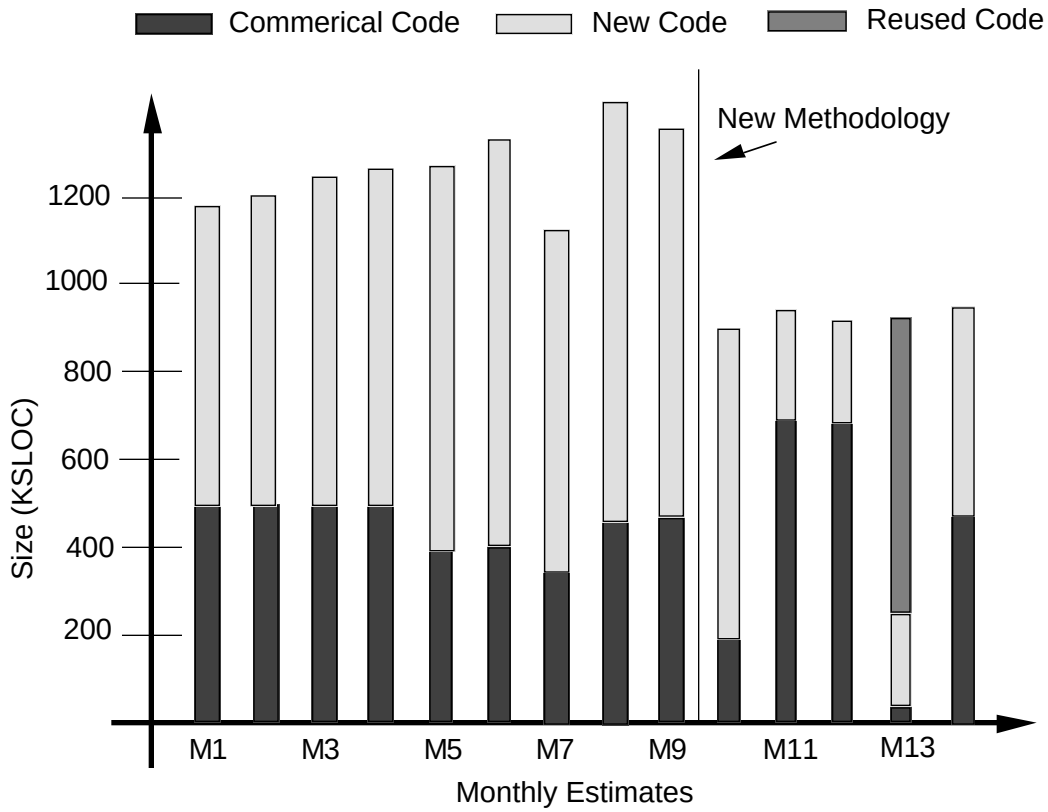

Figure 1: Example Illustrating the Need for Understanding Data

When determining what data to collect, the preferred data is that which is a direct result of the developer's process. For example, count the number of computer software units to be developed by looking in the preliminary design documentation. Limit the use of data that is below the granularity supported by the organization's software process. For example, when collecting software defect data and accompanying process information about the defects, use

the process already in place. To do otherwise might not be cost effective and may adversely affect the productivity and schedule of the contract. If the organization's process does not support a useful level of granularity, then the capability of management and their decisions will also limited.

> "…You cannot just use numbers to control things. The numbers must properly represent the process being controlled, and they must be sufficiently well defined and verified to provide a reliable basis for action. While process measurements are essential for orderly improvement, careful planning and preparation are required or the results are likely to be disappointing." [Humphrey]

The manager must realize that most issues will require multiple (and seemingly unrelated) data items to characterize the issues. For example, if the issue is the amount of rework and how it is affecting the project's progress, the manager would need to have data on progress, the number of items being reworked, how much effort is being spent on rework, how effective the process is at finding items that need rework early, etc. If an issue is how the amount of rework is affecting the budget or end quality of the product, a different set of data items might be collected. The data needed to characterize an issue is also dependent on the development phase (e.g., requirements, design, or coding). Using the rework example, determining what items to use for measuring rework will be different during the requirements phase than during the design phase.

Measurement Data Types and Partitioning

The plans, assumptions, models, and historical data provide the basis for using actual project data. There are four types of data that managers will use:

- Historical
- Plan
- Actual
- Projections

A project will use historical data and assumptions to generate estimates. The estimates, along with other information and knowledge of the system and environment, are then used as the basis for planning the project. Actual data is collected by the project and used as the basis for comparing the performance to plans and for projecting future trends. The project also uses historical data to determine the realism and achieveability of the plans.

Partitioning a data item can also improve the insight into the issues. For example, data could be partitioned by:

- Processor (i.e., target platforms)
- Language
- Organization or subcontractor

- Computer software configuration item (CSCI), computer software component (CSC), or computer software unit (CSU)
- Configuration (e.g., avionics of two different airplanes)
- Severity
- Incremental build
- Facility or Lab
- Work breakdown structure (WBS) element

The level of partitioning will have an impact on how the data is used and how useful the data will be. For instance, if effort is collected at the organizational level rather than at lower levels of a WBS, resource allocation issues (e.g., certain modules or units not receiving enough testing) may be disguised. The partitions should highlight the area(s) of concern so that trends from the indicators can help the program manager determine if the issues are being mitigated. For another example, SLOC can be partitioned by newly developed, modified, or reused. If only total SLOC is measured, changes to where the SLOC comes from may go undetected. These changes may reflect significant differences in the effort and schedule required to complete the project. In general, the lower the level of detail for the data, the better the probability of isolating problem areas. However, the lower the level of detail, the higher the cost of the measurement process. Therefore, tradeoffs will need to be made.

Whenever data is collected, its time of collection and source should be noted. The data should be treated as proprietary to prevent abuses and reduced availability and to alleviate the concerns about how the data will be used. People will already be concerned about the use of the data and using data as a tool to quantify their performance. If measurement becomes an adversarial issue, the quality and availability of the data will probably be greatly reduced. Therefore, the data should be used as a tool for improving communications and as a basis for identifying and resolving issues.

Along with the source and time, assumptions and other knowledge about the data should also be noted. Later, when analyzing the data, the project manager will need the data and all the information available about the data to completely understand the information. For example, if actual effort expended is lower than expected, there might have been uncontrollable problems such as staff awaiting security clearances or waiting for equipment to be delivered. Other types of information could signal potential problems, e.g., changes to plans.

Assistance for Defining Software Measures

The SEI has developed frameworks to assist a project in identifying its issues and ensuring that its measurement definitions are crafted in such as way that the data collected is sufficient in nature to provide insight into the issues. The frameworks are based on a series of checklists that a project uses simultaneously for issue identification and measurement definition. Checklists that are available, and the characteristics they can be used to address, are shown in table 1 below.

Unit of Measure	Characteristics Addressed
Physical source lines of code Logical source lines of code	Size, reuse, rework
Staff hours	Effort, cost, resource allocations
Calendar dates for process milestones Calendar dates for deliverables	Schedule, progress
Problems and defects	Quality, improvement trends, rework, readiness for delivery

Table 1: SEI Core Measures

In preparing the frameworks, the SEI was guided by two criteria:

- Communication: If someone uses a checklist or definition to record and report measurement results, will others know precisely what is included, what is excluded, and how the measurement unit is defined?
- Repeatability: Would others be able to repeat the measurements and get the same results?

These properties are important if misunderstandings or misinterpretations are to be avoided. They are essential if consistency is to be achieved across projects or from organization to organization.

The frameworks permit managers to identify multiple data attributes to address project issues. Although, managers would like to have a single number to characterize their programs, experienced managers search for answers using the insight that multiple measures and data attributes provide. At the organizational level, collecting data with multiple attributes also separates projects into characteristics that can then be used at a later date for comparisons such as benchmarking. For example, separating lines of code data by language, production method, and type of system.

The Size Checklist

The size checklist can be used to help define counts of physical and logical source lines of code. There are attributes for separating by development status (e.g., estimated or planned) and also for separating the data by language. Other attributes include:

Statement type: e.g., executable, declarations, or comments

How produced: e.g., programmed, modified, removed, or generated

Origin: e.g., new work or prior work from a previous version or library

Usage: e.g., part of the primary product or external to the primary product

Delivery: e.g., delivered as source or executable or non delivered, but under configuration control

Functionality: e.g., operative or inoperative such as nonfunctional code

Replications: e.g., copies inserted during compile, originals only, post development replicates

A portion of page one of the size checklists is shown in Figure 2.

Definition Checklist for Source Statement Counts

Physical Source Lines of Code Date: ***7/9/92***

(basic definition) ***SEI***

Measurement unit: **Physical source lines** ✔ / **Logical source statements**

Statement type	**Definition** ✔	**Data array**		**Includes**	**Excludes**
When a line or statement contains more than one type, classify it as the type with the highest precedence.					
1 Executable	**Order of precedence ->**		1	✔	
2 Nonexecutable					
3 Declarations			2	✔	
4 Compiler directives			3	✔	
5 Comments					
6 On their own lines			4		✔
7			5		✔
8 Banners and nonblank spacers			6		✔
9 Blank (empty) comments			7		✔
10 Blank lines			8		✔
12					

How produced	**Definition** ✔	**Data array**	**Includes**	**Excludes**
1 Programmed			✔	
2 Generated with source code generators			✔	
3 Converted with automated translators			✔	
4 Copied or reused without change			✔	
5 Modified			✔	
6 Removed				✔
7				
8				

Figure 2: Portion of SEI Checklist for Defining Lines of Code

The Effort Measures Checklist

The Effort checklist can be used to help define counts of physical and logical source lines of code. There are attributes for:

Type of labor: e.g., direct and indirect

Hour information: e.g., regular time, overtime, salaried, or hourly

Employment class: e.g., full time, part time, contract, or consultant

Labor class: e.g., management, programmer, or support staff

Activity: e.g., development or maintenance

Product-level functions: e.g., system level functions, build level functions, Design, coding, or testing

A portion of page two of the effort checklists is illustrated in Figure 3.

Product-Level Functions CSCI-Level Functions (Major Functional Element)	Totals include	Totals exclude	Report totals
Software requirements analysis		✔	
Design			
Preliminary design	✔		
Detailed design	✔		
Code & development testing			
Code & unit testing	✔		
Function (CSC) integration and testing	✔		
CSCI integration & testing	✔		
IV&V	✔		✔
Management	✔		
Software quality assurance		✔	
Configuration management		✔	
Documentation	✔		✔
Rework			
Software requirements		✔	
Software implementation	✔		
Re-design	✔		
Re-coding	✔		
Re-testing	✔		
Documentation	✔		

Figure 3: Portion of SEI Checklist for Defining Software Effort

The Problem Count Checklist

The problem count checklist can be used to help define counts of defects and enhancements regarding software products. There are attributes for:

Problem status: e.g., open and closed

Problem type: e.g., software defects, hardware problems, new requirements, or enhancements

Uniqueness: e.g., original or duplicates

Criticality: e.g., how critical is the problem regarding the operation of the system

Urgency: e.g., how urgent is the problem to the customer reporting it

Finding attribute: e.g., design, code, inspections, reviews, or testing

A portion of page one of the problem count checklist is shown in Figure 4.

Problem Count Definition Checklist-1				
Software Product ID [ ]				
Definition Identifier: [ ]			**Date: [ ]**	
Attributes/Values	**Definition [ ]**		**Specification [ ]**	
Problem Status	**Include**	**Exclude**	**Value Count**	**Array Count**
Open	✔		✔	
Recognized				
Evaluated				✔
Resolved				✔
Closed	✔		✔	
Problem Type	**Include**	**Exclude**	**Value Count**	**Array Count**
Software Defect				
Requirements Defect	✔		✔	
Design Defect	✔		✔	
Code Defect	✔		✔	
Operational Document Defect	✔		✔	
Test Case Defect		✔		
Other Work Product Defect		✔		
Other Problems				
Hardware Problem		✔		
Operating/System Software		✔		
User Mistake		✔		
Operations Mistake		✔		
New Requirement/Enhancement		✔		
Undetermined				
Not Repeatable/Cause Unkown		✔		
Value Not Identified		✔		

Figure 4: Portion of SEI Checklist for Defining Problems and Defects

The Schedule Measures Checklists

The schedule measures checklists allows the project to determine what milestones and deliverables it will plan and track progress towards. In the schedule checklists there are capabilities to separate the items to be tracked by builds and overall. For each item to be tracked, there is the capability to define exit criteria on that item and to further decompose that item to track key events regarding that item. For example, when sign-off by the customer, management, and/or quality assurance is obtained.

As part of the schedule series of checklists, the ability to track schedule and progress by counting completed work units is also included.

Conclusion

In summary, the objective is to have an understandable set of measurement data that can be processed into different reports and indicators to address and help answer different issues and questions as they occur. To accomplish this, projects should collect much of the data itself and at a low enough level so that the data can be clearly understood and represented in many different ways. To assist in identifying the issues and defining data, the SEI has developed frameworks for common software measures. Examples of those frameworks was illustrated. See [Park] for more information on counting lines of code; , see [Goethert] for information on counting effort or tracking schedules; and, see [Florac] for information on counting problems and defects.

Bibliography

[Boehm] Boehm, B., *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[Demarco] DeMarco, T., *Controlling Software Projects*. New York, New York: Yourdan Press, 1982.

[Deming] Deming, W. E., "*The New Economics for Education, Government, and Industry,*" Quality Enhancement Seminars, Inc., Los Angeles, CA, 1992.

[Florac] Florac, William, A., *Software Quality Measurement: A Framework for Counting Problems and Defects*, CMU/SEI-92-TR-22, Software Engineering Institute - Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1992.

[Goethert] Goethert, Wolfhart, B., et. al., *Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information*, CMU/SEI-92-TR-21, Software Engineering Institute - Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1992.

[Humphrey] Humphrey, W. S., *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley Publishing Co., 1989.

[Juran] The Juran Institute, "The Tools of Quality; Part V: Check Lists," *Quality Progress*, October 1990.

[Park] Park, Robert, E., *Software Size Measurement: A Framework for Counting Source Statements*, CMU/SEI-92-TR-20, Software Engineering Institute - Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1992.

[Rozum92] Rozum, James A., *Software Measurement Concepts for Acquisition Program Managers*, CMU/SEI-92-TR-11, Software Engineering Institute - Carnegie Mellon University, Pittsburgh, Pennsylvania, June 1992.

[Rozum94] Rozum, James A. and William A. Florac, A DoD Software Measurement Pilot: Applying the SEI Core Measures, CMU/SEI-94-TR-16, Software Engineering Institute - Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1995.

[Shewhart] Shewhart, W. A., *Statistical Method From the Viewpoint of Quality Control*, The Graduate School - The Department of Agriculture, Washington D.C., 1939.